

ETAPE 1: Jee WebApp, CRUD / BDD

Objectif

Arriver à monter une application Web Jee comprenant:

- un formulaire
- agissant sur une unique table de BDD (table "Abonnés", par exemple: 4 champs Prénom, Nom, Âge, et une clé primaire) . Une seule table dans la BDD.
- Le formulaire doit permettre les opérations CRUD sur les enregistrements de la table de la BDD.

BOM:

- Tomcat,
- mariaDB,
- client HeidiSQL <https://www.heidisql.com/> (client graphique SQL)
- accès "management" BDD avec le user "lauriane/lauriane"
- accès BDD par l'application Web Java Jee avec l'utilisateur:
 - username: "appli-de-lauriane"
 - mot de passe: "mdp@ppli-1@urian3"

Autres installations (hors VM) :

- eclipse sur mon pc/mac habituel ("hôte" de virt.).
- HeidiSQL

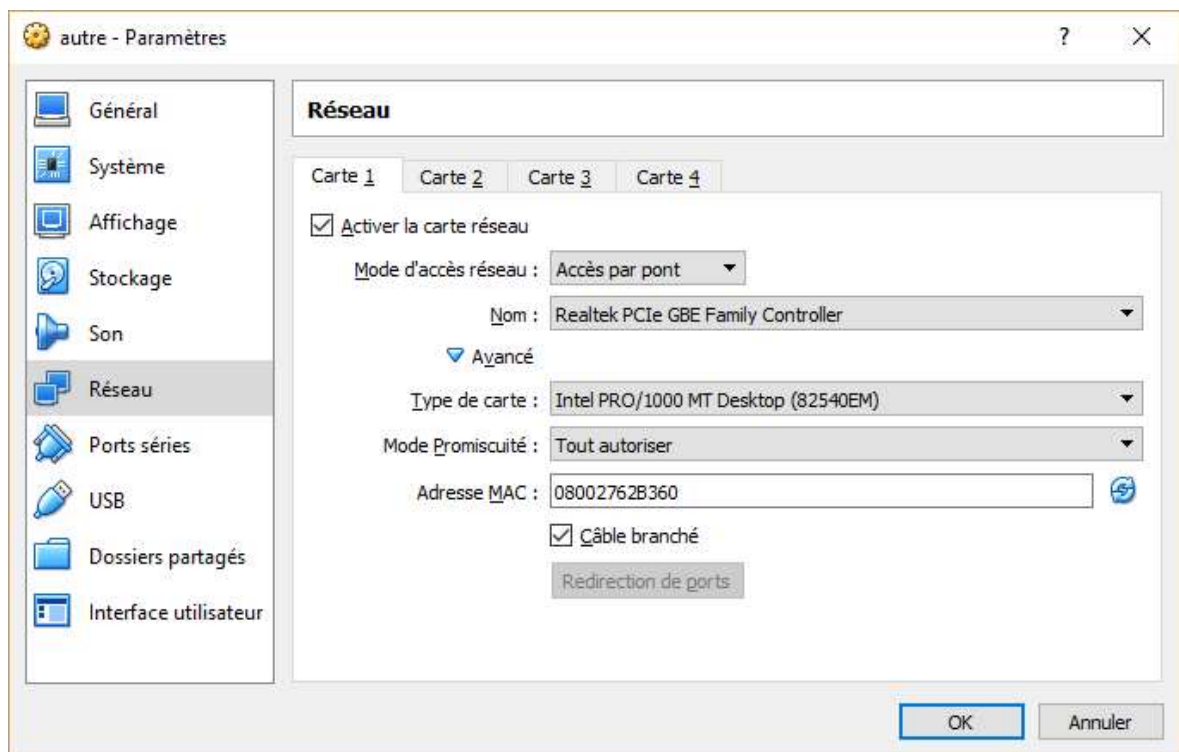
Mode d'emploi:

- Nous allons créer une nouvelle VM Ubuntu. Pour pouvoir réaliser l'ensemble des opérations suivantes, la VM Ubuntu que tu vas créer:
 - doit avoir accès à Internet.

(ou à un réseau IP, et dans ce réseau pouvoir accéder par le protocole TCP/IP, à ce que l'on appelle des "repository" ("dépôt") valides pour le logiciel "apt-get", mais ceci est une autre histoire).

Un test simple: si dans ta VM Ubuntu, tu as accès à internet en ouvrant firefox, et que tu connaît l'adresse IP de ta "box" FAI, alors tu as ce qu'il faut.

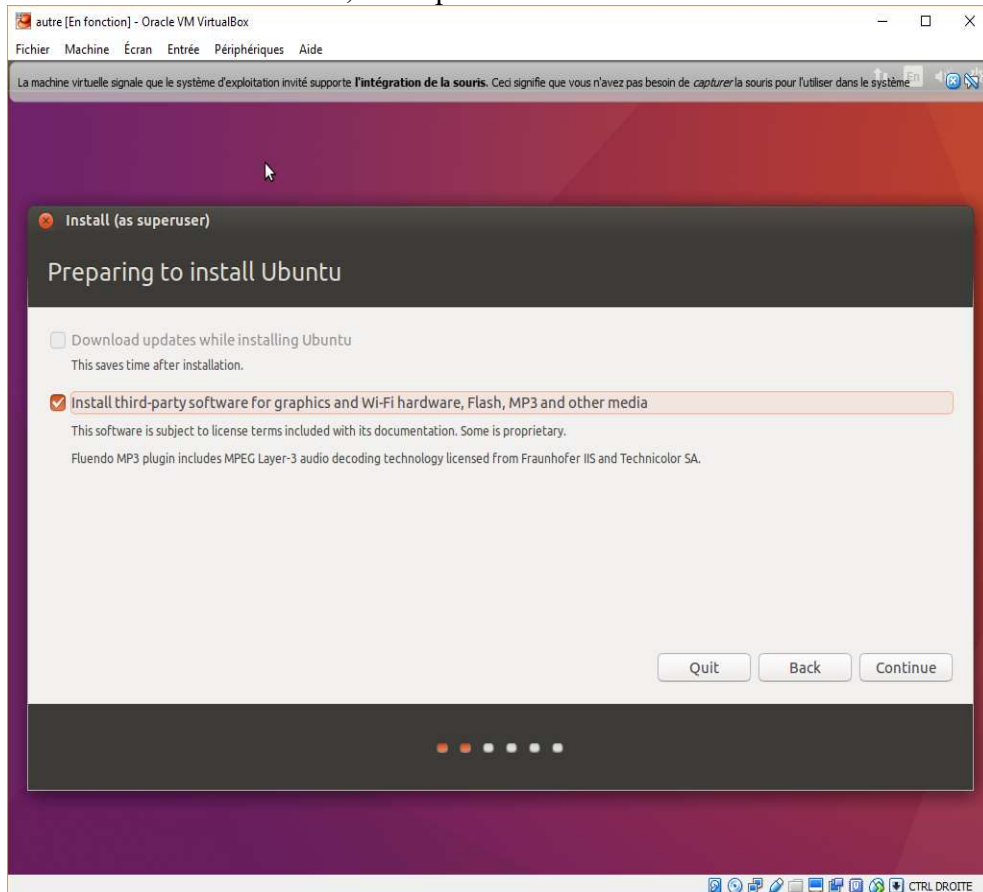
- Crées une nouvelle VM: tu y installeras donc Ubuntu. La carte réseau virtuelle de la VM doit être configurée :
 - avec un **Mode d'accès réseau** de type "Bridge Network" ("Accès par pont"),
 - et un **Mode Promiscuité** de type "Allow all" ("Tout autoriser"):



Ci-dessus, on voit que j'ai configuré le mode d'accès réseau "Accès par pont", et le mode promiscuité "Tout autoriser" pour l'une des cartes réseau virtuelles, d'une VM Virtual Box que j'ai créée

- Tu devra installer une version bien précise d'Ubuntu, la version "16.04 LTS". Tu peux télécharger une image ISO d'installation de cette version à l'aide du lien suivant (copies-colle ce lien dans la barre d'adresse Firefox):
<ftp://ftp.free.fr/mirrors/ftp.ubuntu.com/releases/16.04/ubuntu-16.04.3-desktop-amd64.iso>
- Si avec le PC/MAC avec lequel tu travailles, tu es connectée à internet via wifi, alors,

pendant l'installation d'Ubuntu, à l'étape suivante de l'installation:



Tu devras cocher l'option "Install third-party software for graphics and Wifi hardware [...]": lorsque cette option est cochée, un pilote ("driver") de carte WIFI sera installé, ce qui permettra à ta VM de se connecter en Wifi.

À une étape ultérieure de l'installation, tu devras créer un utilisateur linux. Tu choisiras pour cet utilisateur le nom "lauriane-lx-usr" et le mot de passe "lemdp".

- Effectue maintenant l'installation Ubuntu.
- Quand l'installation est terminée, si ton PC/MAC est connecté en WIFI à internet, il faut configurer la connexion Wifi de ta nouvelle VM. Pour t'aider dans cette configuration, reportes-toi à l' Annexe I. "Configuration WIFI de la VM".
- Lorsque l'installation Ubuntu est terminée, et que tu as accès à Internet dans ta VM, ouvres un terminal, et exécutes les commandes:

```
sudo apt-get install -y git
git clone https://github.com/Jean-Baptiste-Lasselle/lauriane
# optionnellement, pour une version spécifique
# (exemple: la release "v2.0")
# git checkout tags/v2.0
# accessoirement, il serait logique de créer d'abord une branche:
# git branch mabranchedetravail
# pour ensuite faire le checkout dans la branche
# git checkout tags/v2.0 -b mabranchedetravail
# afin de pouvoir faire le merge plus tard...
```

- puis les commandes:

```
sudo chmod +x lauriane/monter-cible-deploiement.sh
sudo lauriane/monter-cible-deploiement.sh
```

- À cette étape, tu as un serveur Tomcat est prêt à être utilisé. Tu peux maintenant déployer une application Web Java Jee exemple. Pour cela ouvres un seond terminal (ou une session SSH avec PUTTY), et exécute la commande:

```
sudo lauriane/deployer-appli-web.sh
```

- pour déployer "un *.war quelconque":

```
NOM_FICHER_WAR=./lauriane/nom-de-ton-fichier.war
sudo lauriane/deployer-appli-web.sh $NOM_FICHER_WAR
```

ANNEXE II. Que la lumière soit

```
# =====>>> le grand principe <<< =====
# =====>>> le grand principe <<< =====

# ==>> donc 2 versionning:
#
#      == [VERSIONNING DE LA RECETTE] un pour le code de la recette de construction de la cible de déploiement
#      == [VERSIONNING DE LA DEPENDANCE] un repo git versionnant:
#
#      ++ fichier docker-compose.yml,
#
#      ++ les fichiers dockerfiles et fichiers permettant de construire (avec un docker build par exemple) l'image
#      customisée de chaque composant de l'infrastructure:
#
#      - [COMPOSANT SGBDR] (construire l'image mariadb avec un fichier de conf custom, permettant de changer le numéro de port):
#
#      le fichier dockerfile:
#
#      FROM mariadb
#      ADD ./mon.fichier.de.conf.custom
#      RUN cp ./mon.fichier.de.conf.custom /etc/mysql/my.cnf
#      CMD #là je crois c'est hérité du FROM
#
#      le fichier de conf custom mariadb: "./mon.fichier.de.conf.custom"
#
# - [COMPOSANT TOMCAT] (construire l'image mariadb avec un fichier de conf custom, permettant de changer le numéro de port)
#
#      et en fait, le code de la recette de déploiement effectue un checkout d'une version bien donnée du fichier docker-compose.yml
#      (et c'est là qu'est le lien entre le numéro de version de la recette, et le numéro de version de la dépendance que constitue le fichier
# [docker-compose.yml])
#
# ==>> donc, en réalité, 1 repo git de versionning de la recette , et N repo git de chaque dépendance (de degré 1) de la recette.
#
# -----
# le lien aux principes "the devops program":
# -----
#
#      == une dépendance de degré zéro, cest: moi-même.
#      == une dépendance de degré zéro de la recette, cest le code source de la recette elle-même.
#
#      == une dépendance de degré zéro d'une appli java, c'est le code source de l'appli java elle-même.
#      == une autre dépendance de degré zéro d'une appli java, c'est le code source/config du build de l'appli java.
#
# -----
# !!!!!!!!!!!!!!! et des roues imbriquées, petite roue dans grande roue, il y a une dimension supplémentaire avec les multiples repos GIT:
# -----
#
# Donc, dans l'arbre des dépendances, pour une dépendance de degré N:
# - le versionning du code source de la dépendance de degré N, mentionne le lien aux versions des dépendances de degré N+1
# - le versionning du code source de la dépendance de degré N, permet de changer le numéro de version d'une dépendance de degré N+1
# !!!!!!! par exemple, pour changer le numéro de port de mariadb de la cible de déploiement
# !!!!!!! (par config de mariadb et / ou changement de mapping docker numéro de port du conteneur docker ) :
#
#      ## ON DETRUIT LA CIBLE DE DEPLOIEMENT
#      on fait docker-compose down
#      ## ON EDITE DU CODE SOURCE
#      on fait une nouvelle version du fichier "docker-compose.yml" en changeant dans ce fichier le mapping du
#      numéro de port dans le et/ou le numéro de version checkouté pour le checkout du repo versionnant le
#      dockerfile pour construire l'image (FROM mariadb ADD ./mon.fichier.de.conf.custom RUN cp ./mon.fichier.de.conf.custom /etc/mysql/my.cnf)
#      on édite sed le fichier "./docker-compose.yml", pour changer le mapping docker du numéro de port d'écoute de mariadb avec un numéro de
#      port exposé par le conteneur du composant SGBDR
#      on édite sed le fichier de conf custom mariadb "./mon.fichier.de.conf.custom", pour changer le numéro de
#      port d'écoute de mariadb ) "à l'intérieur du conteneur"
#
#      ## ON COMMIT l'un ou l'autre, ou les deux...
#      [git add "./docker-compose.yml"]
#      [git commit -m "nouveau mapping docker numéro de port mariadb -p noportexterne=noportinterne"]
#      [git add "./mon.fichier.de.conf.custom"]
#      [git commit -m "nouveau numéro de port mariadb noportinterne="]
#
#      ## ON TAGGUE
#      on détruit le tag ETAT_INITIAL_CIBLE_DEPLOIEMENT : [git tag -d vETAT_INITIAL_CIBLE_DEPLOIEMENT] (san détruire le commit associé au tag détruit)
#      on git taggue le commit [git tag -a vETAT_INITIAL_CIBLE_DEPLOIEMENT -m "ETAT_INITIAL_CIBLE_DEPLOIEMENT: nouveau numéro de port etc..."]
#      on fait un git docker-compose up
#      on fait docker-compose up
#
#      ## IL MANQUE LA PARTIE DOCKER BUILDS AVEC DOCKER COMMIT POUR VERSIONNER IMAGES MONTEES ...?
#
#
# le grand principe: IL MANQUE LA PARTIE DOCKER BUILDS AVEC DOCKER COMMIT POUR VERSIONNER
# IMAGES MONTEES
# mais pour moi on pourrait ne pas changer les numeros de version
```

ANNEXE I. Configuration WIFI de la VM

- Pour ce faire, tu as besoin:
 - Du nom de la connexion Wifi: tu dois le connaître, car pour connecter ton PC/MAC en Wifi, tu as du rechercher le "nom de la connexion Wifi". Exemple de nom de connexion Wifi: "Livebox-3818". Plus précisément, c'est l'identifiant du point d'accès Wifi que tu utilises.

- Et si la connexion Wifi n'est pas libre, d'autres informations, par exemple un mot de passe (c'est souvent le cas dans des lieux ouverts au public comme les bar et bibliothèques). Je vais supposer qu'un mot de passe est suffisant pour être autorisé à se connecter en Wifi, et si ce n'est pas le cas....: débrouilles-toi pour connecter ta VM en gros comme tu l'as fait pour connecter ton PC / MAC.

Tu disposes donc de l'identifiant (le nom) du point d'accès Wifi, et du mot de passe. Pour configurer ton wifi:

1. exécutes la commande:

```
sudo gedit /etc/network/interfaces
```

2. Regardes le contenu de ce fichier, tu devrais trouver un bloc qui ressemble à ceci (ici, j'ai mis "wlan0, mais ce pourrait être "wlan1", "wlan2", "wlan3" ...):

```
auto wlan0
iface wlan0 inet dhcp
```

3. il y a peut-être deux lignes supplémentaires sous ce bloc (si elle n'y sont pas, ajoutes-les):

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid etquelquechoslà
wpa-psk etautrechoseici
```

4. édites le fichier pour insérer l'identifiant (le nom) du point d'accès Wifi, et le mot de passe comme ci-dessous:

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid l-identifiant-du-point-d-acces-wifi
wpa-psk et-le-mot-de-passe-ici
```

5. Nota Bene: les accès WIFI libres existent encore à certains endroits, et dans ce cas, n'ajoutes pas la dernière ligne (précisant le mot de passe):

```
auto wlan0
iface wlan0 inet dhcp
wpa-ssid l-identifiant-du-point-d-acces-wifi
```

6. Enregistres (Ctrl +S), quitte gedit (Ctrl + Q), puis exécutes dans un terminal:

```
ip addr flush wlan0
systemctl restart networking.service
```

Ta VM devrait maintenant avoir accès à internet. Si ce n'est pas le cas, alors.. Eh bien tu vas te débrouiller, mais voici quelque liens qui pourraient t'aider:

<https://doc.ubuntu-fr.org/wifi>

https://doc.ubuntu-fr.org/wifi_ligne_de_commande

Bon courage! ;)

ANNEXE III. Configuration d'un data source pour applications Jee

• Pour Tomcat

- Pour le cas datasource au niveau du serveur:
 - [docker cp dans le conteneur] vérifier la présence de `$CATALINA_HOME/lib/tomcat-dbc.jar`, le déployer si nécessaire
 - [docker cp dans le conteneur] déployer dans `$CATALINA_HOME/lib` le jar du driver JDBC
 - [docker cp dans le conteneur] configuration à appliquer dans le `[$TOMCAT_HOME/conf/context.xml]` (à vérifier par META-INF/context.xml du projet):

vierge, ce fichier a pour contenu:

```
<?xml version='1.0' encoding='utf-8'?>
<!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor
      license agreements. See the NOTICE file distributed with this work for additional
      information regarding copyright ownership. The ASF licenses this file to
      You under the Apache License, Version 2.0 (the "License"); you may not use
      this file except in compliance with the License. You may obtain a copy of
      the license at http://www.apache.org/licenses/LICENSE-2.0 Unless required
      by applicable law or agreed to in writing, software distributed under the
      License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
      OF ANY KIND, either express or implied. See the License for the specific
      language governing permissions and limitations under the License. -->
<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources. If one of these changes, the -->
    <!-- web application will be reloaded. -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!-- <Manager pathname="" /> -->

    <!-- Uncomment this to enable Comet connection tacking (provides events
         on session expiration as well as webapp lifecycle) -->
    <!-- <Valve className="org.apache.catalina.valves.CometConnectionManagerValve"
         /> -->

</Context>
```

et on y ajoute la balise <Resource>:

```
<?xml version='1.0' encoding='utf-8'?>
<!-- Licensed to the Apache Software Foundation (ASF) under one or more contributor
      license agreements. See the NOTICE file distributed with this work for additional
      information regarding copyright ownership. The ASF licenses this file to
      You under the Apache License, Version 2.0 (the "License"); you may not use
      this file except in compliance with the License. You may obtain a copy of
      the license at http://www.apache.org/licenses/LICENSE-2.0 Unless required
      by applicable law or agreed to in writing, software distributed under the
      License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
      OF ANY KIND, either express or implied. See the License for the specific
      language governing permissions and limitations under the License. -->
<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources. If one of these changes, the -->
    <!-- web application will be reloaded. -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!-- <Manager pathname="" /> -->

    <!-- Uncomment this to enable Comet connection tacking (provides events
         on session expiration as well as webapp lifecycle) -->
    <!-- <Valve className="org.apache.catalina.valves.CometConnectionManagerValve"
         /> -->
    <!-- Configuration du datasource -->

    <Resource name="organisation/SourceDeDonnees" auth="Container" type="javax.sql.DataSource"
              maxActive="20"
              maxIdle="10"
              maxWait="10000"
              username="lauriane"
              password="lauriane"
              driverClassName="org.mariadb.jdbc.Driver"
              url="jdbc:mariadb://localhost:8456/bdd_oraganisaction"
              />

</Context>
```

et pour finir [docker restart du conteneur]

[Donc il me faut comme paramètres supplémentaires du plugin]

Ci-dessous, une correspondance entre les balises d'un pom.xml utilisant le plugin, et les variables

d'environnement utilisées dans les scripts de déploiement du datasource:

- `MARIADB_JDBC_DRIVER_CLASS_NAME=`
`<jdbc-driver-classname>org.mariadb.jdbc.Driver</jdbc-driver-classname>`
 - `JEE_DATASOURCE_NAME=`
`<jee-datasource-name>organisation/SourceDeDonnees</jee-datasource-name>`
 - `JEE_DATASOURCE_AUTH_USERNAME=<lx-user>lauriane</lx-user>`
 - `JEE_DATASOURCE_AUTH_USERPWD=<lx-pwd>lauriane</lx-pwd>`
 - `JEE_DATASOURCE_MAX_ACTIVE=`
`<jee-datasource-max-active>20</jee-datasource-max-active>`
 - `JEE_DATASOURCE_MAX_IDLE=`
`<jee-datasource-max-idle>10</jee-datasource-max-idle>`
 - `JEE_DATASOURCE_MAX_WAIT=`
`<jee-datasource-max-wait>10000</jee-datasource-max-wait>`
 - `JEE_DATASOURCE_URL_PREFIX=`
`<!-- ici il y a le numéro de port et adresse IP SGBDR-->`
`<jee-datasource-url-prefix>jdbc:mariadb</jee-datasource-url-prefix>`
 - `JEE_DATASOURCE_URL=`
`<!-- Non, inutile dans la configuration, l'URL peut`
`être formée à partir:`
 - de l'adresse IP utilisée par le SGBDR
 - du numéro de port utilisé par le SGBDR
 - de la valeur de `$JEE_DATASOURCE_URL_PREFIX` `<jdbc-datasource-url-prefix>`
 - de la valeur de `$NOM_BDD_APPLI` `<jdbc-datasource-url-prefix>``-->`
`<jee-datasource-url>jdbc:mariadb://192.168.1.149:3306/db</jee-datasource-url>`
- dans [WEB-INF/web.xml], ajouter une référence au datasource configuré dans le [TOMCAT_HOME/conf/context.xml] ccc

```
<resource-ref>
  <description>DB Connection</description>
  <res-ref-name>jdbc/TestDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

- Enfin, pour tester le datasource, il faut:
- créer une table de tests "MembresAssos" dans la BDD, et y mettre quelques enregistrements. Exemple de table:

MembresAssos

prenom

nom

username

email

age

-

- Pour le cas datasource au niveau de l'application:

// à préciser, mais:

Donc j'ai quasiment finis d'automatiser le déploiement du datasource pour l'application
L'erreur sur laquelle je tombe, indique entre autre une erreur de compatibilité entre la version de JRE,
et le driver MariaDB/JDBC:

```
lauriane@lauriane-vm: ~  
.ContainerBase.addChildInternal ContainerBase.addChild: start:  
org.apache.catalina.LifecycleException: Failed to start component [StandardEngi  
ne[Catalina].StandardHost[localhost].StandardContext[/host-manager]]  
    at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:162)  
    at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase  
.java:753)  
    at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:72  
9)  
    at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:717)  
    at org.apache.catalina.startup.HostConfig.deployDirectory(HostConfig.jav  
a:1126)  
    at org.apache.catalina.startup.HostConfig$DeployDirectory.run(HostConfig  
.java:1868)  
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:47  
3)  
    at java.util.concurrent.FutureTask.run(FutureTask.java:262)  
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.  
java:1152)  
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor  
.java:622)  
    at java.lang.Thread.run(Thread.java:748)  
Caused by: java.lang.UnsupportedClassVersionError: org/mariadb/jdbc/Driver : Uns  
upported major.minor version 52.0  
    at java.lang.ClassLoader.defineClass1(Native Method)  
    at java.lang.ClassLoader.defineClass(ClassLoader.java:803)  
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:14  
2)  
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:442)  
    at java.net.URLClassLoader.access$100(URLClassLoader.java:64)  
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java --version"  
Unrecognized option: --version  
Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.  
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java -version"  
java version "1.7.0_151"  
OpenJDK Runtime Environment (IcedTea 2.6.11) (7u151-2.6.11-2~deb8u1)  
OpenJDK 64-Bit Server VM (build 24.151-b01, mixed mode)  
lauriane@lauriane-vm:~$
```

Ci-dessus, ce sont les logs de tomcat dans le conteneur, et on voit une ligne :

"Caused by: java.lang.UnsupportedClassVersionError: org/maraidb/jdbc/Driver : Unsupported
major.minor version 52.0"

Hors, le code 52 est associé au JDK 8 pour ce type d'exceptions natives Java, et elle est émise par la
classe Pilote JDBC du pilote MariaDB/JDBC:

The issue is because of Java version mismatch. Referring to the Wikipedia Java Class Reference :

- J2SE 9 = 53
- J2SE 8 = 52
- J2SE 7 = 51
- J2SE 6.0 = 50
- J2SE 5.0 = 49
- JDK 1.4 = 48
- JDK 1.3 = 47
- JDK 1.2 = 46
- JDK 1.1 = 45

These are the assigned major numbers. The error regarding the unsupported major version is because **during compile time you are using a higher JDK and a lower JDK during runtime.**

Thus, the 'major.minor version 52.0' error is possibly because the jar was compiled in jdk 1.8, but you are trying to run it using a jdk 1.7 environment. The reported number is the **required** number, not the number you are using. To solve this, it's always better to have the jdk and jre pointed to the same version.

In IntelliJ,

- Go to Maven Settings -> Maven -> Importing. Set the JDK for importer to 1.8
- Go to Maven Settings -> Maven -> Runner. Set the JRE to 1.8
- Go to File -> Project Structure -> SDKs. Make sure the JDK home path is set to 1.8

Restart IntelliJ

Another approach which might help is by instructing IDEA which JDK version to start up with.

Go to : `Applications/IntelliJ IDEA 15 CE.app/Contents/Info.plist` and replace the JVM version with :

```
<key>VMVersion</key>  
<string>1.8</string>
```

edited May 25 '17 at 11:23 answered Mar 8 '16 at 11:22

Join Stack Overflow to learn, share knowledge, and build your career. Email Sign Up or Sign Up with Google Facebook

Bon, et de plus on voit que la version de java dans le conteneur qui exécute tomcat est:

```
lauriane@lauriane-vm: ~  
ContainerBase.addChildInternal ContainerBase.addChild: start:  
org.apache.catalina.LifecycleException: Failed to start component [StandardEngine[Catalina].StandardHost[localhost].StandardContext[/host-manager]]  
    at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:162)  
    at org.apache.catalina.core.ContainerBase.addChildInternal(ContainerBase.java:753)  
    at org.apache.catalina.core.ContainerBase.addChild(ContainerBase.java:729)  
    at org.apache.catalina.core.StandardHost.addChild(StandardHost.java:717)  
    at org.apache.catalina.startup.HostConfig.deployDirectory(HostConfig.java:1126)  
    at org.apache.catalina.startup.HostConfig$DeployDirectory.run(HostConfig.java:1868)  
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:473)  
    at java.util.concurrent.FutureTask.run(FutureTask.java:262)  
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1152)  
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:622)  
    at java.lang.Thread.run(Thread.java:748)  
Caused by: java.lang.UnsupportedClassVersionError: org/mariadb/jdbc/Driver : Unsupported major.minor version 52.0  
    at java.lang.ClassLoader.defineClass1(Native Method)  
    at java.lang.ClassLoader.defineClass(ClassLoader.java:803)  
    at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)  
    at java.net.URLClassLoader.defineClass(URLClassLoader.java:442)  
    at java.net.URLClassLoader.access$100(URLClassLoader.java:64)  
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java --version"  
Unrecognized option: --version  
Error: Could not create the Java Virtual Machine.  
Error: A fatal exception has occurred. Program will exit.  
lauriane@lauriane-vm:~$ sudo docker exec -it ciblededeploiement-composant-srv-jee /bin/bash -c "java -version"  
java version "1.7.0_151"  
OpenJDK Runtime Environment (IcedTea 2.6.11) (7u151-2.6.11-2~deb8u1)  
OpenJDK 64-Bit Server VM (build 24.151-b01, mixed mode)  
lauriane@lauriane-vm:~$
```

Donc il va falloir que je customise le dockerfile de la construction de tomcat. Dans ce dockerfile, je ferai l'installation de la JRE 8 au lieu de la JRE 7.

la dc officielle mariadb JDBC confirme l'incompatibilité (sachant que c'est la version 2.2.1 que je déploierais pour le test) :

The screenshot shows the MariaDB Connector/J website. The main content area is titled "Choosing a Version" and provides information about driver versions compatible with MariaDB servers. It lists current maintenance versions (2.2 for Java 8 and 9, 1.7 for Java 6 and 7) and notes that versions before 1.2.0 are no longer supported. A table titled "Requirements" shows the compatibility between driver versions and Java versions. The table has two columns: "Driver version" and "Java version".

Driver version	Java version
2.x	Java 9, Java 8
> 1.6.1 - < 2	Java 9, Java 8, Java 7, Java 6
1.2 - 1.6.1	Java 8, Java 7,
1.1	Java 8, Java 7, Java 6

Below the table, it states that JNA (net.java.dev.jna:jna) and JNA-PLATFORM (net.java.dev.jna:jna-platform) >4.2.1 is needed when connecting to the server with unix sockets or windows pipes. The "Source Code" section provides a link to the GitHub repository and a command to clone it.

J'ai eut une autre déclinaison de mes problèmes d'incompatibilité avec la version de JRE dans le conteneur docker:

mes propres classes étaient compilées pour une cible JRE 8 (avec une source en Java 8), et ça re-levait une exception d'incompatibilité (et sans aucun log des exceptions de la webapp dans les logs serveurs):

J'ai testé ça en écrivant une simple Servlet faisant une réponse par une page HTML: en l'appelant directement avec son `<url-pattern>`, j'ai obtenu cette erreur d'incompatibilité entre la version de JRE, et la version de JDK que j'ai utilisé pour compiler mon code, avec les deux paramètres (source/target). Bref, j'ai changé la configuration du "maven-compiler-plugin", pour que mon code soit compilé en 1,7 en source, et en 1,7 en target, et le problème a été résolu, ma Servlet fonctionne (et permet de gérer les exceptions survenues dans l'application).

Arrivé là, j'ai modifié mon application pour qu'elle fasse un traitement des exceptions par config `<error-page>` dans le web.xml. Les exceptions sont attrapées par une Servlet, qui fait l'affichage d'autant d'infos que possible à propos de l'exception survenue. Une page jsp de test des exceptions permet de vérifier qu'une `NullPointerException` Levée, est bien traitée par ma servlet [{@see GestionnaireDexceptions }](#). Tandis qu'aucune exception n'est attrapée lorsque j'invoque la page faisant usage de mon datasource configuré dans le web.xml

Oui, là, j'ai un problème de complexité pour mes tests:

version du driver jdbc /mariadb

version de tomcat

version de dbcp

version de Java dans le conteneur

version de Java en source et target de compilation (pom.xml)

fichier de configuration \$CATALINA_BASE/conf/context.xml

fichier de configuration \$CATALINA_BASE/conf/server.xml

fichier de configuration \$MONPROJET/WEB-INF/context.xml

Si j'ai seulement 2 versions de chaque, j'ai un gros nombre de combinaisons à tester

- **Pour Wildfly**